Санкт-Петербургский государственный университет

Направление математика и механика

Прикладная математика и информатика

Жигалова Алена Валерьевна

Генерация случайных многоугольников

Бакалаврская работа

Научный руководитель:

ст. преп. Зациорский А. С.

Рецензент:

к. ф.- м. н., доцент Бухвалова В. В.

Санкт-Петербург

2017

Saint Petersburg State University
Main Field of Study Mathematics and Mechanics
Area of Specialisation Applied Mathematics and Computer Science

Zhigalova Alena

Generating random polygons

Bachelor's Thesis

Scientific Supervisor:
Senior Lecturer Zatsiorskiy A.

Reviewer:
PhD, Associate Professor Bukhvalova V.

Saint-Petersburg
2017

Оглавление

Введение

Существуют различные области применения, в которых основными объектами являются многоугольники. Например, среди этих областей есть алгоритмы для задач вычислительной геометрии, алгоритмы компьютерной графики, графические приложения, которые стремятся имитировать природные структуры: облака, земельные образования, биологические или гидрологические явления.

Генерация случайных многоугольников используется для проверки правильности и оценки потребления процессорного времени алгоритмов, работающих на многоугольниках. Данные могут быть заданы вручную или случайно. Главной сложностью генерации вручную является потребность предвидеть особенности задачи, чтобы за минимиальное время протестировать исключительные случаи. Сгенерированные случайным образом данные позволяют увеличить объем и статистическую выборку работы алгоритма, также при большом количестве испытаний будут обнаружены все имеющиеся в алгоритме ошибки.

Данные для статистического тестирования должны быть разнообразны и достаточны по количеству, так как нет смысла проверять только на частных случаях многоугольников и на маленькой выборке. Часто практически невозможно получить достаточно большое количество разнообразных входных данных. Тогда наилучший вариант — запустить алгоритм для достаточно большого числа случайных входных данных. А так как в каждой задаче объекты могут быть своими, то хорошо бы задавать параметры, чтобы генерировать подходящие для этой задачи данные.

В данной работе рассматривается задача генерации случайных многоугольников с заданным количеством вершин и заданным классом (выпуклый, звездный). Описанные выше параметры являются обязательными и выбираются при построении многоугольника. Также есть и дополнительные: максимальная и минимальная длины ребра, диапазон изменения угла и дисперсия величины угла. Дополнительные параметры не являются необходимыми в поставленной задаче и учитываются при возможности разрабатываемых алгоритмов.

Работа состоит из двух глав. В первой главе рассматриваются основные известные алгоритмы и идеи на которых они основаны, анализируется применимость к поставленной задаче. Во второй главе описываются и сравниваются между собой алгоритмы, разработанные в рамках данной работы.

1. Существующие алгоритмы генерации случайных многоугольников

Генерация случайных геометрических объектов являлась предметом интереса многих ученых. К примеру, Epstein [5] изучал случайную генерацию триангуляций. Zhu et al.[8] презентовал алгоритм для случайной генерации x-монотонного многоугольника с заданным множеством вершин. Эвристики для генерации простого многоугольника были иследованы O'Rouke Virmani [7].

Основными же алгоритмами, близкими к поставленной задаче являются:

1.1. Алгоритм генерации выпуклого случайного многоугольника

Алгоритм $rcpoly(T, a, b, \theta)$, представленный Никулиным Е.А. [2], генерирует выпуклый случайный многоугольник. Входными данными являются:

- T — первая вершина
- a и b — границы длины ребер
- θ — максимальное приращение углов

Схема действия алгоритма:

- p_1 в исходной точке T, $\phi_1 = 0$
  $\phi_{n+1} = \phi_n + \Delta\phi_n$, где $\Delta\phi_n, = rnd(\theta), 90 \leq \theta \leq 180$
  $r_n = a + rnd(b - a)$
- Точка $q = p_n + r_n[\cos(\phi_n), \sin(\phi_n)]$ тестируется с $n = 2$:
  - левее прямой $\{T, p_2\}$
  - правее прямой $\{p_n, T\}$
  - левее прямой $\{p_{n-1}, p_n\}$

  Если все условия выполнены, то $p_{n+1} = q$.

Данный алгоритм не позволяет решить поставленную задачу, так как многоугольники, получаемые в результате, могут иметь различное количество вершин.

1.2. Алгоритм RPA генерации случайного многоугольника

В работе [6] приведён алгоритм, который генерирует случайные многоугольники из случайно сгенерированных точек за $O(n^2 \log n)$.

Схема действия алгоритма:

- Генерируются 3 неколлиниарные точки и рисуется многоугольник размера 3.
- Следующая точка добавляется к P данным образом:

  n-3 раз:

  1. Выбрать случайный отрезок из P с концами в V_a и V_b.
  2. Выбрать видимую область P. Область видимая для V_a и V_b (а именно достижима линией без пересечения любых существующих линий с двумя конечными точками (V_a и V_b) должна найтись.

     Каждый отрезок P исследуется, чтобы определить, видима ли часть P для V_a и V_b. Видимая часть добавляется к P'. Предполагается, что вершины хранятся в формате отсортированного связанного списка, а $V.next$ и $V.prev$ доступны.

     Эта область может рассмотрена как многоугольник P'.
  3. Случайно выбираем новую точку V_c, принадлежащую P', путем деления P' на треугольники.
  4. Добавить V_c в P' с помощью отрезков (V_a, V_c) и (V_b, V_c). P становится P'.

Данный алгоритм строит произвольные n-угольники, однако он не позволяет учитывать их класс. Многоугольники могут получится любого класса и пользователь не может это регулировать.

1.3. Алгоритм Никулина генерации случайного многоугольника

Алгоритм $arpoly(T, a, b, m, M)$, представленный Никулиным Е.А. [1], для создания случайного многоугольника основан на методе имитации процесса дискретного броуновского движения на плоскости в случайных направлениях со случайными длинами шагов.

Данный алгоритм также строит случайные n–угольники и не позволяет контролировать класс получаемого многоугольника. Однако он учитывается специальные входные параметры.

На входе алгоритма заданы:

- T задает расположение начальной вершины p_1.
- следующая пара аргументов a и b определяет диапазон изменения длины шага $d \in (a, b)$.
- четвертый аргумент $m \geq 3$ задает минимальный номер вершины p_m, с которой начинается проверка возможности замыкания полилинии ребром $p_m p_1$.
- пятый аргумент $M \geq m$ ограничивает максимальное число шагов хаотического блуждания для предотвращения его зацикливания в случае невозможности замыкания полилинии из глубин построенного лабиринта.

Схема действия алгоритма:

- Следущая вершина p_{n+1} строится через проведение от точки p_n отрезка случайной длины d под углом $\phi = rnd(2\pi)$. На конце получается пробная точка q. Точка q тестируется на пересечение $p_n q$ и $p_k p_{k+1}$, если есть пересечение, то генерация q повторяется.
- При $n \geq M$ проверяем на позможность замыкания многоугольник.
- Если за M итераций $p_1 p_2 ... p_M$ не может замкнуться без самопересечений, то заново начинаем строить многоугольник с точки T.

2. Разработанные алгоритмы генерации случайных многоугольников

Определение 1. *Многоугольник называется* ***выпуклым****, если все его точки лежат по одну сторону от любой прямой, проходящей через две его соседние вершины.*

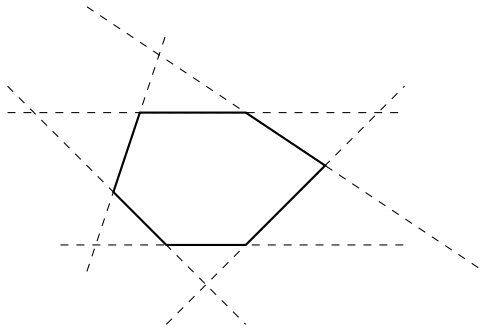


Рис. 1. Выпуклый многоугольник

2.1. Выпуклые многоугольники: первый алгоритм

Входные данные алгоритма:

- n — количество вершин.
- Границы диапазона допустимых длин ребёр (`maxlength` и `minlength`).

Схема действия алгоритма:

1. Обработка данных, генерация первой точки
   а. Генерируется угол первой точки:
   `angle = rnd (0, 360)`.
   б. Точка `O` называется центром, вокруг неё будет строиться многоугольник.
   `m = minlength / (2*sin ((180/n) * (`π`/180))` .
   `M = maxlength / (2*sin((180/n) * (`π`/180))` .
   Данные формулы позволяют выбрать максимальное `M` и минимальное `m` расстояние от `O` точек многоугольника с учётом диапазона границы длин ребёр. Преобразования основаны на формуле нахождения стороны основания в равнобедренном треугольнике (`a = b / (2 * sin (`β`/2))`) и формуле перевода из радиан в градусы (``π / 180`).

в. Расстояние от центра общее для всех вершин выбирается случайным образом из обозначенного диапазона:
`length = rnd (m, M)`

2. Вычисление координат для остальных точек i.

а. Для всех i генерируется отклонение от стандартного угла:
ϕ`[i] = random (0, (360 / n)) - (360 / n) / 2` .

б. Каждая следущая точка i генерируется следущим образом:
`x = O.x + cos (angle+(360/n)*i+`ϕ`[i])*`π`/180)*length` .
`y = O.y + sin (angle+(360/n)*i+`ϕ`[i])*`π`/180)*length` .

Данный алгоритм генерирует не все выпуклые многоугольники, а только определенной формы. Так как окружность разделяется на секторы в пределах которых находится вершина, и поэтому при больших n многоугольники начинают стремиться к форме окружности.

Однако алгоритм имеет только один цикл длины n и его вычислительная сложность $O(n)$, что позволяет строить многоугольники данной формы быстро.

2.2. Выпуклые многоугольники: второй алгоритм

На входе алгоритма задано n — количество вершин.
eps - малое число, необходимое для генератора случайных вещественных чисел.

1. Построение лучей и центральной точки:

- O — точка, вокруг которой строится многоугольник, она же и является центром координат.
- Вычисляются углы α_i между вершинами, таким образом чтобы $\sum \alpha_i = 2 * \pi$ и $\alpha_i < \pi$.
  Данная генерация реализована следущим образом:
  $\gamma_0 = 0$
  В качестве γ_i считается угол между OS_i и OS_0.
  Для $i < n - 2$:
  Каждый угол строится с учетом следущих ограничений:

– Если $\gamma_{i-1} + \pi \geq 2 * \pi$, тогда γ_i, чтобы не превысить $2 * \pi$, строится в диапазоне $[\gamma_{i-1} + eps, 2 * \pi - eps]$.

После построения угол проверяется на то, что есть достаточно места для следущих углов: если $(2 * \pi - \gamma_i)/(n - i) < eps$, то оставшегося места недостаточно и угол генерируется снова.

– Если $\gamma_{i-1} + \pi < 2 * \pi$, тогда γ_i, чтобы не превысить π, строится в диапазоне $[\gamma_{i-1} + eps, \gamma_{i-1} + \pi - eps]$.

Так же происходит снова проверка на то, что есть достаточно места для следущих углов: если $(2 * \pi - \gamma_i)/(n - i) < eps$, то оставшегося места недостаточно и угол генерируется снова.

Для $i = n - 1$, т.е для предпоследнего угла (последний угол получается автоматически). Существует проверка на то, чтобы последний угол был $< \pi$, то есть должно быть $2 * \pi - \gamma_i \leq \pi$

- Из углов γ_i вычисляются углы α_i:

  $\alpha_0 = \gamma_1$.
  Для $i < n - 1 : \alpha_i = \gamma_i - \sum_{k=0}^{i-1} \alpha_k$.
  $\alpha_{n-1} = 2 * \pi - \gamma_{n-2}$.
  Из точки O проводятся лучи OS_i с углами α_i между OS_i и OS_{i+1}.

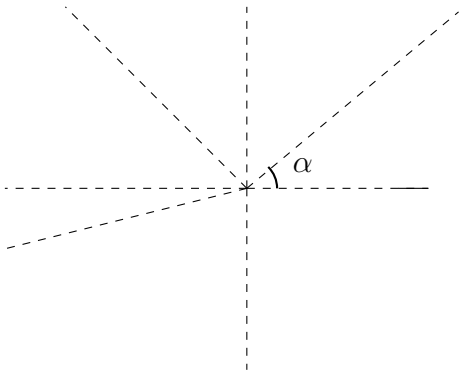


Рис. 2. точка O и лучи

2. Генерация точки A_1:

- На OS_0 выбирается случайная точка A_0.
- $\angle\beta = rnd(0 + eps, \pi - eps)$.

- Через A_0 проводится прямая под углом β к A_0O до пересечения с OS_1 — это точка A_1.

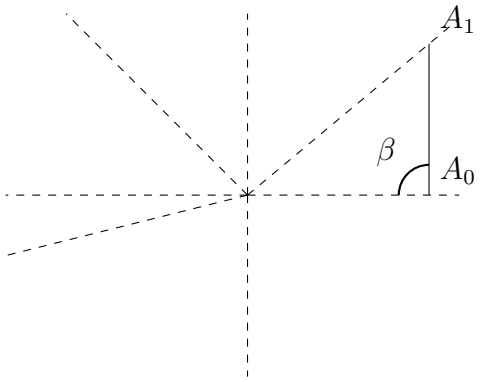


Рис. 3. точки A_0 и A_1

3. Генерация точки $A_i, 2 \leq i \leq n$:

   $\angle\beta = (lowerb, upperb)$

   Через A_{i-1} проводится прямая под углом β к $A_{i-1}O$ до пересечения с OS_i — это точка A_i.

Верхняя граница $upperb = \min(\pi - \angle OA_{i-1}A_{i-2}, \angle\theta, \angle OA_{i-1}F)$, рассмотрим случаи, при которых важно каждое из ограничений.

Верхняя граница $upperb = \pi - \angle OA_{i-1}A_{i-2}$ для $\angle\beta$ позволяет контролировать невозможность совершения правого поворота, который бы нарушал выпуклость между точками A_i, A_{i-1}, A_{i-2}.

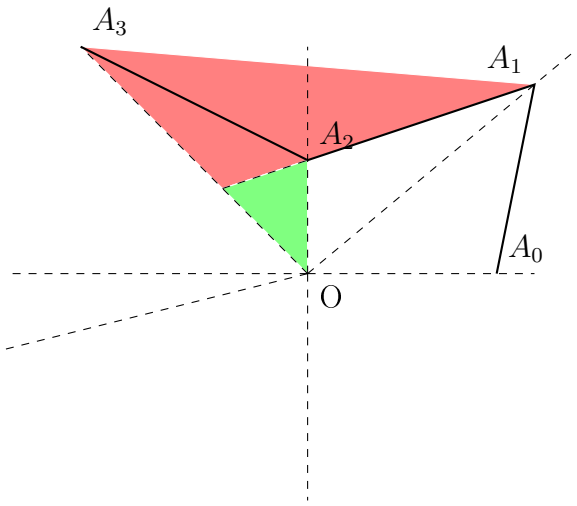


Рис. 4. Правый поворот: нарушение верхней границы $\angle\beta$

На рисунке 4 можно увидеть как совершение правого поворота для точки A_3 нарушает выпуклость многоугольника. Зеленым цветом обозначена допустимая область, а красным область, где граница будет нарушена.

При некоторых β строящийся луч из A_{i-1} может не пересекать луч OS_i. Подобный случай представлен на рисунке 5. В данном случае в качестве верхней границы максимальный угол $\angle\theta$, при котором будет возможно пересечение A_2A_3 и OS_3.

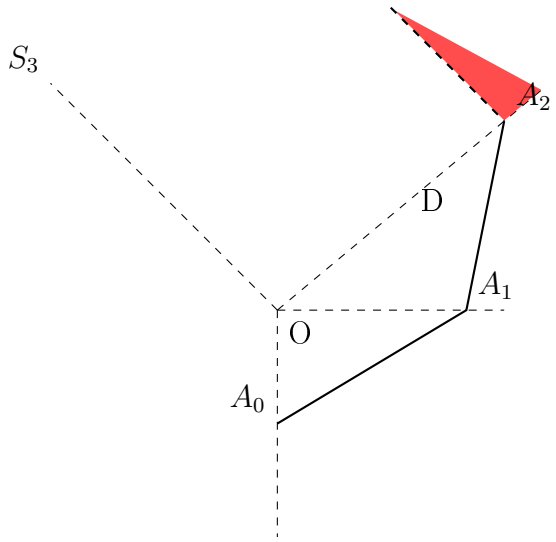


Рис. 5. Верхняя граница $\angle\beta$

Однако из-за необходимости замыкать ломанную существует еще одно ограничение на верхнюю границу $\angle\beta$. Продолжим A_1A_0 до пересечения с OS_i — пусть это точка F. $\angle\beta$ не может превышать $\angle OA_{i-1}F$. Нарушение этого правила при выборе точки A_4 представлено на рисунке 6. Зеленым цветом обозначена допустимая область, а красным область, где граница будет нарушена.

Таким образом $upperb = \min(\pi - \angle OA_{i-1}A_{i-2}, \angle\theta, \angle OA_{i-1}F)$.

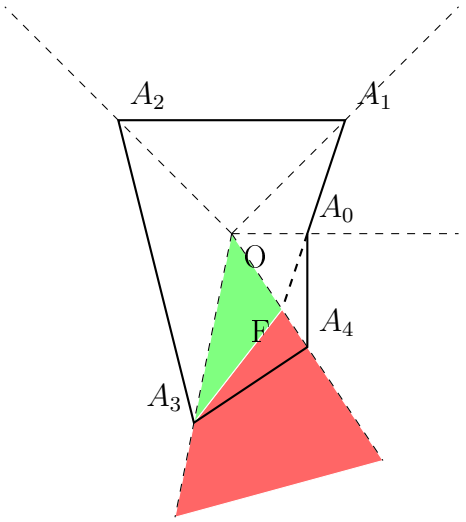


Рис. 6. нарушение $upperb = \angle OA_{i-1}F$

Нижняя граница

$$lowerb = \begin{cases} 0 & \text{если } S_0, S_i \text{ лежат по разные стороны относительно } OS_{i-1} \\ \angle A_{i-1}OA_0 & \text{иначе} \end{cases}$$

Нижняя граница для $\angle\beta$ позволяет обеспечить замыкание ломанной левым поворотом. Нарушение нижней границы при построении A_5 представлено на рисунке 7.

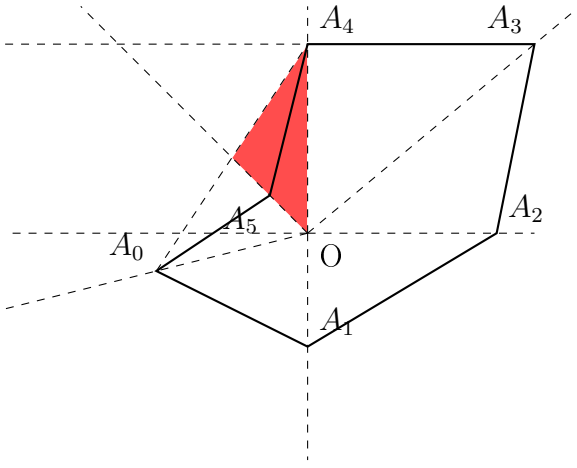


Рис. 7. Нарушение нижней границы для $\angle\beta$

Однако нижняя граница $\angle\beta$ может быть равна 0 в том случае, когда точки лучей S_0 и S_i лежат по разные стороны относительно прямой OS_{i-1}. Для проверка их взаимного расположения воспользуемся псевдоскалярным произведением.

Определение 2. ***Псевдоскалярным*** *или* ***косым произведением векторов*** *на плоскости называется число* $[a, b] = |a||b| \sin\theta$, *где* θ *— угол вращения (против часовой стрелки) от* a *к* b. *Если хотя бы один из векторов* a *и* b *нулевой, то полагают* $[a, b] = 0$.

Если векторы заданы своими координатами $a(x_1, y_1), b(x_2, y_2)$, *то косое произведение* $[a, b] = x_1y_2 - x_2y_1$.

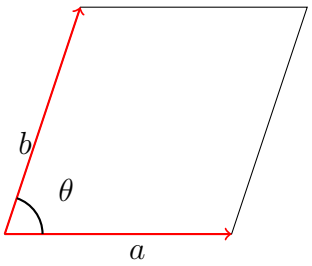
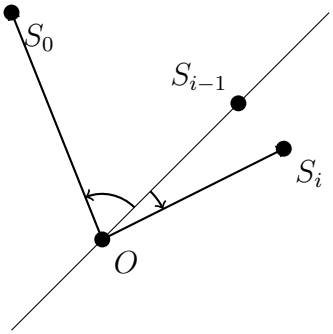


Рис. 8. Косое произведение a и b

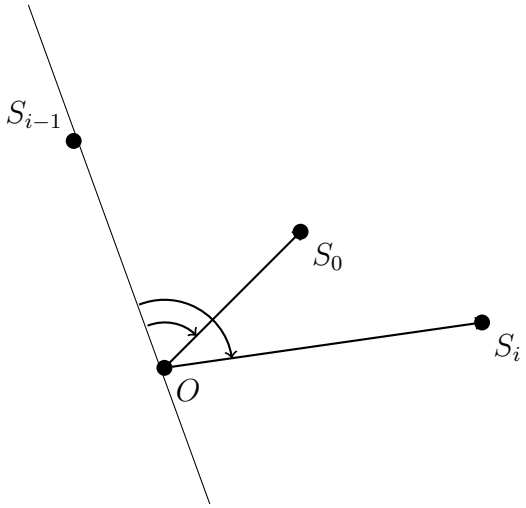
Геометрически косое произведение векторов представляет собой ориентированную площадь параллелограмма, натянутого на эти вектора.

Если точки лежат по разные стороны, то псевдоскалярные произведения разных знаков, и их произведение отрицательно. Если же точки расположены по одну сторону, то косые произведения одного знака, и их произведение положительно.
Возможные случаи:

1. $[OS_{i-1}, OS_0] * [OS_{i-1}, OS_i] < 0$– точки лежат по разные стороны.



2. $[OS_{i-1}, OS_0] * [OS_{i-1}, OS_i] > 0$ — точки лежат по одну сторону.



3. $[OS_{i-1}, OS_0] * [OS_{i-1}, OS_i] = 0$ — одна (или две) из точек лежит на прямой.

Пример когда S_0 и S_i лежат по разные стороны от прямой OS_{i-1} можно увидеть на рисунке 9. Точки S_0 и S_4 лежат по разные стороны прямой OS_3, любой выбор точки A_4 не нарушит выпуклость многоугольника относительно замыкания ломаной.

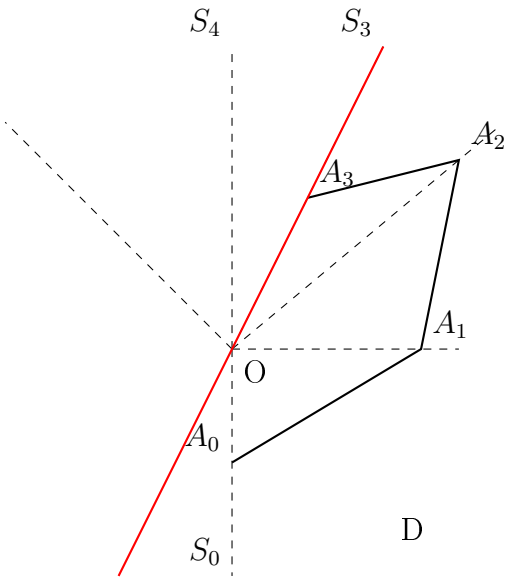


Рис. 9. $lowerb = 0$

Данный алгоритм позволяет нарисовать любой выпуклый случайный n-угольник.

2.3. Тестирование алгоритмов генерации выпуклых многоугольников

Для того чтобы сравнить производительность алгоритмов генерации выпуклых многоугольников, были проведены тесты. Время указано в миллисекундах для построения 10000 экземпляров.

Таблица 1. Сравнение алгоритмов генерации выпуклых многоугольников.

	Среднее время выполнения первого алгоритма	Среднее время выполнения второго алгоритма	Первый алгоритм быстрее второго на
$n = 5$	63 мс	250 мс	297%
$n = 10$	98 мс	483 мс	392%
$n = 20$	164 мс	911 мс	455%

Можно видеть, что среднее время работы второго алгоритма существенно больше чем первого. Как видно из последнего столбца при увеличении числа вершин разница во времени только возрастает.

2.4. Звёздные многоугольники

Определение 3. *Многоугольник P называется* ***звёздным****, если у него существует такая точка z, что для каждой точки p из P отрезок zp целиком лежит внутри P. Множество всех точек z с этим свойством называется* ***ядром*** *P.*

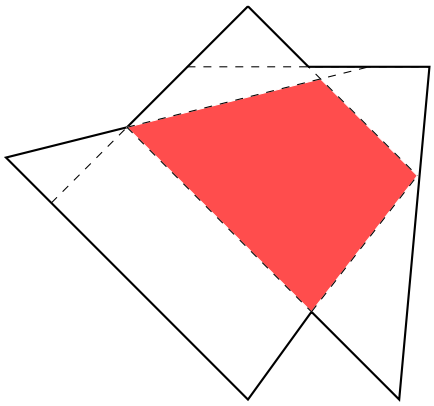


Рис. 10. Звёздный многоугольник

На входе алгоритма задано n - количество вершин.

Схема действия алгоритма:

1. O — точка вокруг которой строится многоугольник.

   Считаются углы α_i между вершинами, каждому присваивается случайное значение, таким образом чтобы $\sum \alpha_i = 2 * \pi$ и $\alpha_i < \pi$.

   Проводятся из O лучи OS_i с углами α_i между OS_i и OS_{i+1}.

2. Выбирается $k = rnd(1, n)$ — это номер луча с которого будет начинаться обход. Происходит перенумерация лучей, чтобы луч OS_k стал OS_0.

   Точка A_0 выбирается случайным образом на OS_0.

   Начинается обход по лучам против часовой стрелки.

   Точка A_1, если это возможно выбирается таким образом, чтобы прямая A_0A_1 и луч OS_2 пересекались. Если точка выбранным таким образом, то для точки A_2 может быть нарушена верхняя граница(соврешен правый поворот), выпуклость многоугольника испорчена и далее точки генерируются случайным образом. Если же описанный выбор A_1 невозможен, то точки A_1 и A_2 многоугольника выбираются на своих лучах случайным образом. И далее, начиная с A_3, алгоритм ищет верхнюю и нижнюю границу для $\angle\beta$ и нарушает одну из них.

- Если есть ограничение на нижнюю границу или верхняя граница ограничена не $\angle\theta$, то граница нарушается и дальнейшие точки генерируются случайно на своих лучах.
- Если же $lowerb = 0$ и $upperb = \angle\theta$, то вершина выбирается случайным образом и следущая вершина проверяется на ограниченность границы. С учетом того, что ломанная должна быть замкнута ограничение на границу обязательно появится на одном из шагов.

Данный алгоритм генерирует случайный невыпуклый звёздный многоугольник.

Также рассмотрим схему действия интуитивного алгоритма генерации звёздного многоугольника:

1. O — точка вокруг которой строится многоугольник.
   Считаются углы α_i между вершинами, каждому присваивается случайное значение, таким образом чтобы $\sum \alpha_i = 2 * \pi$ и $\alpha_i < \pi$.
   Проводятся из O лучи OS_i с углами α_i между OS_i и OS_{i+1}.
2. Начинается обход лучей OS_i против часовой стрелки. На каждом луче OS_i выбирается случайным образом точка A_i.

В результате действия алгоритма получится случайный звёздный, но не обязательно невыпуклый многоугольник. Добавим проверку на выпуклость получившегося многоугольника: у выпуклого многоугольника знаки косых произведений $[A_iA_{i+1}, A_{i+1}A_{i+2}]$ должны быть либо положительны, либо отрицательны.
Начинаем обход против часовой стрелки с вершины A_0, будем считать косые произведения и смотреть на их знаки:

- Как только получим произведения разных знаков завершим проверку, так как исследуемый многоугольник невыпуклый.
- Если же все произведения получились одного знака, то исследуемый многоугольник выпуклый и запустим случайную генерацию с 1 шага, пока не получим невыпуклый многоугольник.

2.5. Тестирование алгоритмов генерации звездных многоугольников

Сравним производительность двух описанных алгоритмов генерации случайных звёздных многоугольников.
Время указано в миллисекундах для построения 10000 экземпляров.

Таблица 2. Сравнение алгоритмов генерации звёздных многоугольников.

	Среднее время выполнения собственного алгоритма	Среднее время выполнения интуитивного алгоритма	Собственный алгоритм быстрее интуитивного на
$n = 5$	223 мс	490 мс	120%
$n = 10$	380 мс	623 мс	64%
$n = 20$	733 мс	1070 мс	46%

Как можно видеть алгоритм, работающий по принципу нарушения границ, выполняется на большом объеме данных быстрее интуитивного алгоритма, строящего многоугольник и проверяющего его на выпуклость. Тем не менее с увеличением числа вершин многоугольника процентная разница во времени уменьшается.
Результаты времени работы для построения существенно меньшего числа экземпляров зависят от удачности выбора интуитивным алгоритмом многоугольника, поэтому при небольшом числе экземляров нельзя сказать какой из алгоритмов работает быстрее.

2.6. Программная реализация

Описанные в предыдущих разделах алгоритмы объединены в один проект, реализованый на языке C#. Общий вид работающей программы представлен на рисунке 11. Код программы размещен по ссылке : https://yadi.sk/d/pstEP6aB3JRUdY/2017. Пользователь должен выбрать количество вершин, класс многоугольника и алгоритм по которому он строится. Также среди дополнительных параметров можно ограничить длину сторон выпуклого многоугольника для первого алгоритма и увидеть лучи, на которых располагаются точки многоугольника для остальных алгоритмов. Координаты вершин многоугольника можно отследить в отдельной области. Картинка сгенерированного многоугольника может быть создана в TEXe с использованием графического пакета Tikz.

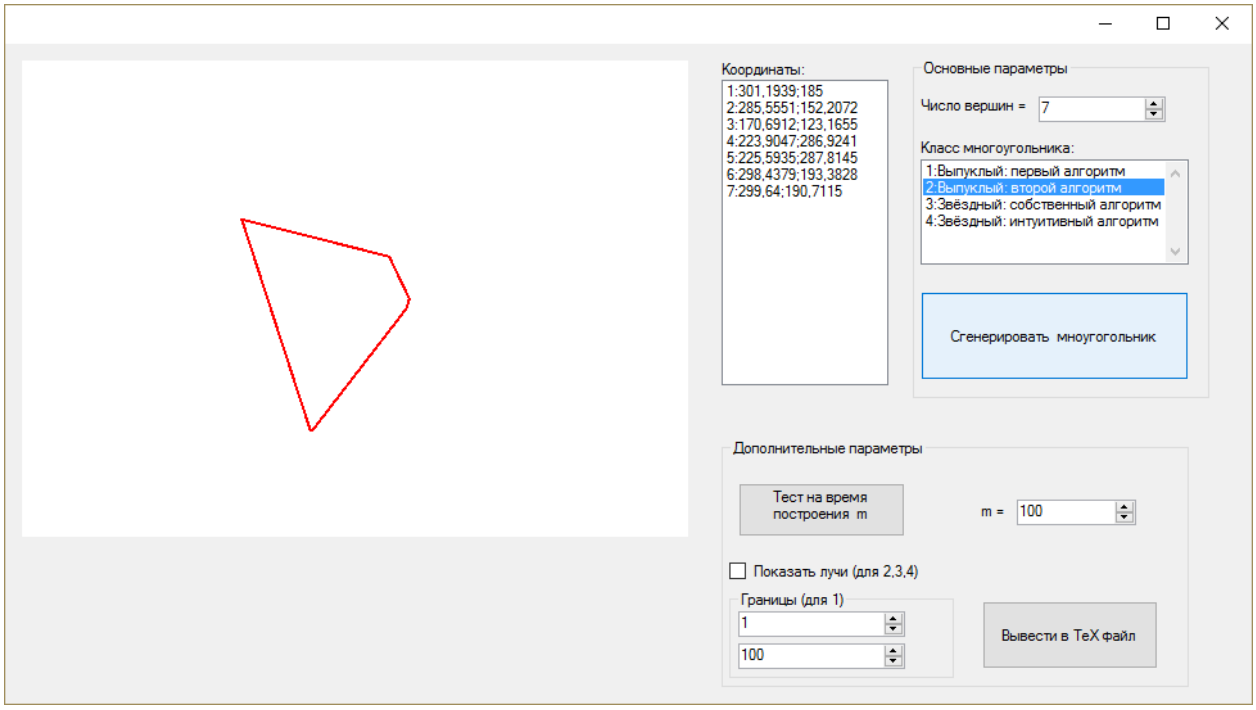


Рис. 11. Результат действия программы

Заключение

В рамках данной работы были разработаны два алгоритма генарации случайных выпуклых многоугольников и алгоритм генерации случайных звёздных многоугольников. Алгоритмы были реализованы объединены в одну общую программу. Программа позволяет кроме основных параметров выбирать частные дополнительные параметры и экспортировать полученный многоугольник в TEX-файл. Также была протестирована, сравнена и проанализирована скорость работы алгоритмов, строящих многоугольники одних типов, на большом объеме экземпляров. Таким образом поставленная задача была выполнена.

В качестве развития темы возможна разработка алгоритмов генерации многоугольников других классов, а так же добавление других дополнительных параметров к разработанным алгоритмам.

Список литературы

1. Никулин Е.А. Труды Нижегородского государственного технического университета им. Р.Е. Алексеева. – НГТУ им. Р.Е. Алексеева, 2013. — С. 84–87.
2. Никулин Е.А. Компьютерная геометрия и алгоритмы машинной графики. – СПб.: БХВ-Петербург, 2003. — С. 51–54.
3. Препарата Ф., Шеймос М. Вычислительная геометрия: Введение. — М.: Мир, 1989. — 560 с.
4. Auer T., Held M. Heuristics for the Generation of Random Polygons. // 8th Canadian Conference on Computational Geometry. — 1996. — P. 38-43.
5. Epstein P. Generating Geometric Objects at Random. // Carleton University. — 1992. — P. 267-274.
6. Dailey D., Whitfield.D. Constructing Random Polygons // Slippery Rock University of Pennsylvania. — 2008. — P. 119–224. .
7. O'Rourke J., Virmani M. Generating Random Polygons. // Smith College. — 1991. — P. 38–44.
8. Zhu C., Sundaram G., Snoeyink J., Mitchell J.S.B. Genrating Random Polygons with Given Vertices // Slippery Rock University of Pennsylvania. — 1996. — P. 277-290.